Conner Jordan

7-25-2023

CST 344 Lab 5

Write Up

**Explain what happens when you run the original program; can you list how many threads are created and what values of i are passed?**

When you run the original program, 10 threads are created in the main() function. The value of i at the time of each thread's creation is passed as an argument to the go() function, but because the address of i (not the value itself) is passed to each thread, all threads actually end up referencing the same memory location where i is stored.

**Do you get the same result if you run it multiple times?**

For me, I was getting the same result (0) each time, I assume this is because of the scheduling behavior on my system. However, the result could vary on different systems or under different conditions.

**What if you are also running some other demanding processes (e.g., compiling a big program, playing a Flash game on a website, or watching streaming video) when you run this program?**

If you are running other demanding processes, the scheduling of your threads might be affected. Depending on how the operating system schedules threads, the created threads may get a chance to run their go() function before the main() thread completes its loop, which could potentially lead to different output.

**The function go() has the parameter arg passed a local variable. Are these variables per-thread or shared state? Where does the compiler store these variables’ states?**

The parameter arg to the function go() is a pointer to a memory location. Each thread receives its own copy of this pointer, so in that sense, arg is a per-thread variable. However, all these pointers are pointing to the same memory location (where i is stored), so the actual value they're referencing (the value of i) is a shared state. These local variables (arg) are typically stored on each thread's stack.

**The main() has local variable i. Is this variable per-thread or shared state? Where does the compiler store this variable?**

The variable i in the main() function is a shared state because its address is passed to each thread, and they all read its value. The variable i is stored on the stack of the main() thread.

**Write down your observations. You probably have seen that there is a bug in the program, where threads may print same values of i. Why?**

The bug in the program is because the address of the variable i (not its value) is passed to each thread. By the time the threads get a chance to execute, the main thread has completed its loop and i has its final value. Since all threads are referencing the same memory location (where i is stored), they all end up printing the same value. This is an example of a race condition in multithreaded programming.